WEEK1 HANDS ON CS

1. Implementation of singleton design.

Code:using System;

public class Square

{

    private static Square instance;

    private int side;

    private **Square**(int side)

    {

*this*.side = side;

    }

    public static Square **GetInstance**(int side)

    {

        if (instance == null)

        {

            instance = new Square(side);

        }

        return instance;

    }

    public int **GetArea**()

    {

        return side \* side;

    }

    public int **GetSide**()

    {

        return side;

    }

    public void **SetSide**(int side)

    {

*this*.side = side;

    }

    public static void **Main**(string[] args)

    {

        Square square1 = Square.**GetInstance**(5);

        Console.**WriteLine**("Square1 side: " + square1.**GetSide**());

        Console.**WriteLine**("Square1 area: " + square1.**GetArea**());

        Square square2 = Square.**GetInstance**(10);

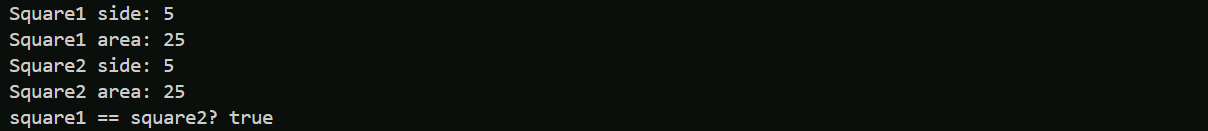
        Console.**WriteLine**("Square2 side: " + square2.**GetSide**());

        Console.**WriteLine**("Square2 area: " + square2.**GetArea**());

        Console.**WriteLine**("square1 == square2? " + (square1 == square2));

    }

}

Output:

1. Implement factory pattern design.

Code:using System;

interface IShape

{

    void **Draw**();

}

class Circle : IShape

{

    public void **Draw**()

    {

        Console.**WriteLine**("Drawing a Circle");

    }

}

class Square : IShape

{

    public void **Draw**()

    {

        Console.**WriteLine**("Drawing a Square");

    }

}

class Rectangle : IShape

{

    public void **Draw**()

    {

        Console.**WriteLine**("Drawing a Rectangle");

    }

}

class ShapeFactory

{

    public IShape **CreateShape**(string type)

    {

        if (type == null)

        {

            return null;

        }

        switch (type.**ToLower**())

        {

            case "circle":

                return new Circle();

            case "square":

                return new Square();

            case "rectangle":

                return new Rectangle();

            default:

                throw new ArgumentException("Unknown shape type: " + type);

        }

    }

}

class FactoryPatternDemo

{

    static void **Main**(string[] args)

    {

        ShapeFactory factory = new ShapeFactory();

        IShape shape1 = factory.**CreateShape**("circle");

        shape1.**Draw**();

        IShape shape2 = factory.**CreateShape**("square");

        shape2.**Draw**();

        IShape shape3 = factory.**CreateShape**("rectangle");

        shape3.**Draw**();

    }

}

Output:



1. E-Commerce System using Data structures and algorithms.

Code:using System;

using System.Collections.Generic;

class Product

{

    public int Id { get; }

    public string Name { get; }

    public double Price { get; }

    public **Product**(int id, string name, double price)

    {

        Id = id;

        Name = name;

        Price = price;

    }

    public void **Display**()

    {

        Console.**WriteLine**($"ID: {Id} | Name: {Name} | Price: ₹{Price:F2}");

    }

}

class CartItem

{

    public Product Product { get; }

    public int Quantity { get; set; }

    public **CartItem**(Product product, int quantity)

    {

        Product = product;

        Quantity = quantity;

    }

    public double **GetTotalPrice**()

    {

        return Product.Price \* Quantity;

    }

    public void **Display**()

    {

        Console.**WriteLine**($"{Product.Name} x{Quantity} = ₹{**GetTotalPrice**():F2}");

    }

}

class Cart

{

    private List<CartItem> items = new List<CartItem>();

    public void **AddItem**(Product product, int quantity)

    {

        foreach (var item in items)

        {

            if (item.Product.Id == product.Id)

            {

                item.Quantity += quantity;

                return;

            }

        }

        items.**Add**(new CartItem(product, quantity));

    }

    public void **ViewCart**()

    {

        if (items.Count == 0)

        {

            Console.**WriteLine**("Cart is empty.");

            return;

        }

        Console.**WriteLine**("Your Cart:");

        foreach (var item in items)

        {

            item.**Display**();

        }

        Console.**WriteLine**($"Total: ₹{**GetTotal**():F2}");

    }

    public double **GetTotal**()

    {

        double total = 0;

        foreach (var item in items)

        {

            total += item.**GetTotalPrice**();

        }

        return total;

    }

    public List<CartItem> **GetItems**()

    {

        return new List<CartItem>(items); *// Return a copy*

    }

    public void **Clear**()

    {

        items.**Clear**();

    }

    public bool **IsEmpty**()

    {

        return items.Count == 0;

    }

}

class Order

{

    private static int orderCount = 0;

    private int orderId;

    private List<CartItem> items;

    private double total;

    public **Order**(List<CartItem> items, double total)

    {

        orderId = ++orderCount;

*this*.items = new List<CartItem>(items);

*this*.total = total;

    }

    public void **Display**()

    {

        Console.**WriteLine**($"Order ID: {orderId}");

        foreach (var item in items)

        {

            item.**Display**();

        }

        Console.**WriteLine**($"Total Amount: ₹{total:F2}");

    }

}

class ECommerceSystem

{

    static void **Main**()

    {

        List<Product> catalog = new List<Product>

        {

            new Product(1, "Samsung M33", 20000),

            new Product(2, "HP Laptop", 50000),

            new Product(3, "Reebok Shoes", 2000),

            new Product(4, "LED TV", 30000)

        };

        Cart cart = new Cart();

        List<Order> orderHistory = new List<Order>();

        while (true)

        {

            Console.**WriteLine**("\n--- E-Commerce System ---");

            Console.**WriteLine**("1. View Products");

            Console.**WriteLine**("2. Add to Cart");

            Console.**WriteLine**("3. View Cart");

            Console.**WriteLine**("4. Place Order");

            Console.**WriteLine**("5. View Orders");

            Console.**WriteLine**("6. Exit");

            Console.**Write**("Choose an option: ");

            string input = Console.**ReadLine**();

            if (!int.**TryParse**(input, out int choice))

            {

                Console.**WriteLine**("Invalid input. Try again.");

                continue;

            }

            switch (choice)

            {

                case 1:

                    Console.**WriteLine**("Available Products:");

                    foreach (var product in catalog)

                    {

                        product.**Display**();

                    }

                    break;

                case 2:

                    Console.**Write**("Enter Product ID: ");

                    int pid = int.**Parse**(Console.**ReadLine**());

                    Console.**Write**("Enter Quantity: ");

                    int qty = int.**Parse**(Console.**ReadLine**());

                    Product selected = catalog.**Find**(p => p.Id == pid);

                    if (selected != null)

                    {

                        cart.**AddItem**(selected, qty);

                        Console.**WriteLine**("Added to cart!");

                    }

                    else

                    {

                        Console.**WriteLine**("Product not found.");

                    }

                    break;

                case 3:

                    cart.**ViewCart**();

                    break;

                case 4:

                    if (cart.**IsEmpty**())

                    {

                        Console.**WriteLine**("Cart is empty.");

                    }

                    else

                    {

                        Order order = new Order(cart.**GetItems**(), cart.**GetTotal**());

                        orderHistory.**Add**(order);

                        Console.**WriteLine**("Order placed successfully!");

                        order.**Display**();

                        cart.**Clear**();

                    }

                    break;

                case 5:

                    if (orderHistory.Count == 0)

                    {

                        Console.**WriteLine**("No orders placed yet.");

                    }

                    else

                    {

                        foreach (var order in orderHistory)

                        {

                            order.**Display**();

                            Console.**WriteLine**("-----");

                        }

                    }

                    break;

                case 6:

                    Console.**WriteLine**("Thank you for shopping!");

                    return;

                default:

                    Console.**WriteLine**("Invalid option. Try again.");

                    break;

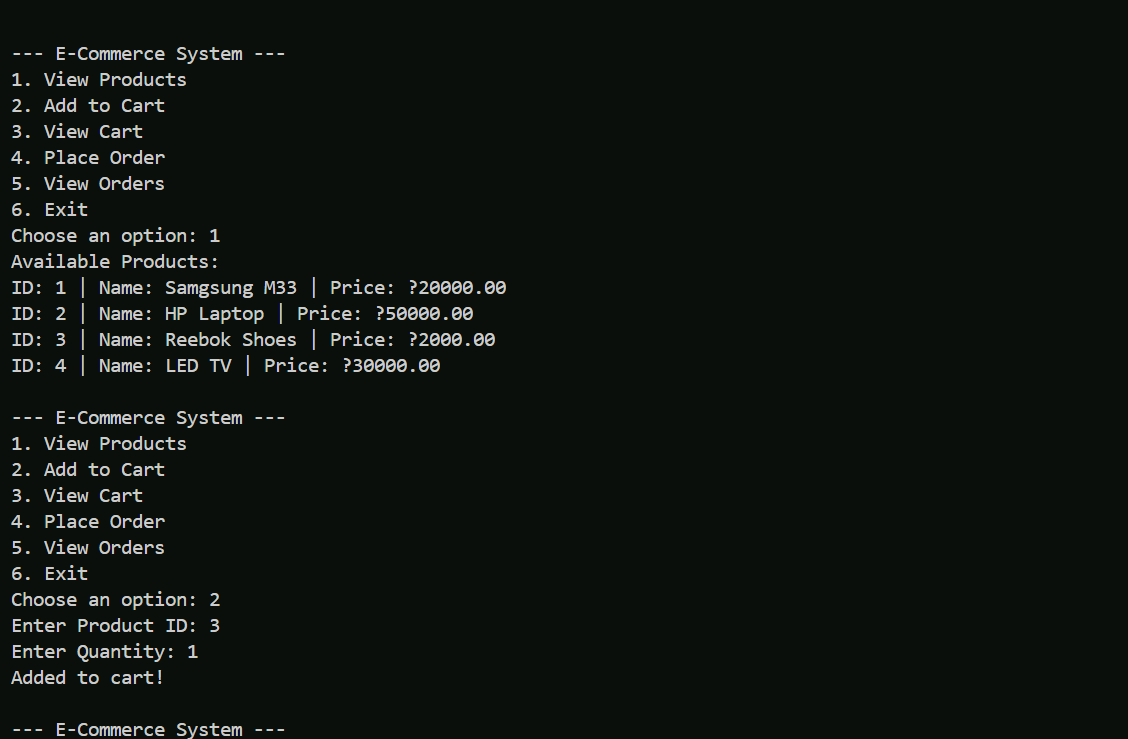
            }

        }

    }

}

Output:



1. Financial Forecasting System using data structure and algorithms.

Code:using System;

using System.Collections.Generic;

class FinancialForecast

{

    private List<double> historicalData;

    public **FinancialForecast**(List<double> historicalData)

    {

*this*.historicalData = historicalData;

    }

    public double **PredictNext**()

    {

        int n = historicalData.Count;

        double sumX = 0, sumY = 0, sumXY = 0, sumXX = 0;

        for (int i = 0; i < n; i++)

        {

            double x = i + 1;

            double y = historicalData[i];

            sumX += x;

            sumY += y;

            sumXY += x \* y;

            sumXX += x \* x;

        }

        double slope = (n \* sumXY - sumX \* sumY) / (n \* sumXX - sumX \* sumX);

        double intercept = (sumY - slope \* sumX) / n;

        double nextX = n + 1;

        return intercept + slope \* nextX;

    }

    static void **Main**()

    {

        List<double> revenues = new List<double> { 10000.0, 12000.0, 12500.0, 15000.0, 16000.0 };

        FinancialForecast forecast = new FinancialForecast(revenues);

        double nextMonthPrediction = forecast.**PredictNext**();

        Console.**WriteLine**($"Predicted revenue for next month: ${nextMonthPrediction:F2}");

    }

}

Output:

